Real time Integrity Control of Operating systems

by

Erik Mellem
Frode Olsen

Masters Thesis in
Information and Communication Technology

Agder University College
Faculty of Engineering and Science

Grimstad, June 2004

Abstract

It can be critical if an intruder gains access to an operating system and modify files. To prevent this Norwegian Defence Research Establishment has proposed a thesis that it is possible to do a real time integrity check of a virtual operating system.

We have looked upon a system using VMware Workstation 4 as the Virtual machine and Tripwire as the integrity controller. We have used Linux in the host operating system and in the virtual operating system.

By modifying the host operating system's IDE driver it is possible to monitor which blocks the virtual operating system is writing. These blocks are then used to find the inode and the path for the file that is written. The file is then integrity checked with Tripwire.

The system we have developed uses approximately 45 seconds from a file is written and until it is discovered that the integrity is violated.

Preface

This master thesis was written as a part of the Norwegian Master degree in Information and Communication Technology. The assignment was provided by the Norwegian Defence Research Establishment, and is a part of the project Computer Network Operation The project has been carried out in the period between January and May 2004.

We would like to thank our supervisors Nils Ulltveit-Moe at Agder University College and Ronny Windvik at FFI, for valuable help and support.

Grimstad, May 2004

Frode Olsen and Erik Mellem

Table of contents

Table of figures

1 Introduction

1.1 Background

Since the beginning of Internet in 1990 the Internet has grown with accelerating speed. More and more computers are connected to this big computer network. As this happens the computers are more vulnerable to attacks and people can steal and modify files. To avoid this firewalls and anti virus software are installed, these security barriers are not complete secure and can be omitted. This can result in a corrupt system where intruders gains access to vital data and use the computer to do more gravely criminality. As a second perimeter of defence integrity control software can be installed. Integrity control discovers unauthorized modification of information. This information can for instance be files on a server. Today the integrity control of a file system is done seldom, probably only a couple of times per week. As a result of this, it can take a long time until the integrity violation is detected. There is also the possibility that an intruder can discover the integrity control and is thereby able to use techniques to bypass it. The project Computers Network Operations at Norwegian Defence Research Establishment needs an integrity control that is able to discover integrity violations of a file system as soon as possible, without being discovered.

1.2 Integrity Control

Integrity control software is used to discover unauthorized modification of information. Integrity control is not meant to deny modification of information, but only to report if there has been any integrity violations. The report is interpreted by a system administrator who checks if it is a false alarm or if an unauthorized modification has occurred. Integrity control is often called the watcher of watchers and is often used to monitor intrusion detection systems and firewalls.

1.3 Hidden structure and virtual machines

To hide an integrity control system from being discovered, it can be placed on a remote computer. The integrity control should be executed as fast as possible after a file is written, and it will thereby need some signalling when a file has been written. This signal must be sent from the monitored system, and it might be intercepted by an intruder who can compromise the integrity control system. A solution to this problem is to use a virtual machine. A virtual machine allows a user to install a virtual operating system inside the original operating system. Both operating systems are then running simultaneously, and it is thereby

possible for the original operating system to monitor the virtual operating system.

1.4 Thesis definition

Integrity control is used to discover unauthorized modification of information. The meaning of integrity control in relation to this project is the process of monitoring which files an intruder changes, and raise an alarm. Today the integrity control of operating systems is rarely performed. Normally this is only done a few times per week. Therefore, it may take a long time before an intruder is detected. There is also a potential danger that the intruder can manipulate the integrity control itself, if it is implemented on the machine being attacked. In this project the integrity control will be done between two operating systems. In the main operating system, called Controller, a virtual machine will run containing the secondary operating system, called Client. The Client will run different network services. It is imperative that the Controller neither can be accessed through the Client nor the network. It is essential that any integrity violations on the Client are detected as quickly as possible after the event happened. An intruder shall not be aware of the existence of a Controller, which is monitoring the Client.

1.5 Our Work

We would like to investigate the possibilities to develop a prototype that is able to detect integrity violations of a file system as soon as possible. FFI has suggested that we look upon the Linux operating system and VMware Workstation [10] as the virtual machine software. We must therefore study how the Linux file systems are build up and how VMware represent the Client's file system. This study will be based on available documentation, and the Linux source code.

To achieve a real time integrity control, the Controller must be aware of which files the Client writes to disk. We must thereby investigate how the Client behaves when files are written to disk, and how the Controller may intercept these write calls.

We must also do a research on how the Client can be isolated from the Controller, and how the Client may be available to the network, while the Controller is not.

We believe that the literature study will take a long time in this project, since neither of us has in-depth knowledge of Linux nor file systems.

2 File system

2.1 EXT3 file system

The EXT3 file system is the most common file system for Linux, developed by Stephen Tweedie. It is an extension of the EXT2 file system with journaling. In case of a system failure, the EXT3 file system is recovered faster than the EXT2. Because it is not necessary to run a file system check to find out which part of the file system that eventually are inconsistent. The EXT3 file system is built up by inodes [01], which works as pointers to where the file physically is located on the disk. Each file has at least one inode. The inode contains information about the file, such as size, when the file was created, modified or deleted. The inode also contains the block numbers which the file occupies.
The EXT3 file system is divided into groups. The size of the groups is dependent of the block size. The size of one block is optional, but a common size is 4 Kilo bytes. If the block size is 1 Kilo byte, each group consists of 8192 blocks [02]. And in a system with a block size of 4 Kilo bytes, each group consists of 32768 blocks. The first block on an EXT3 partition is the superblock. The superblock contains information about the file system, such as group size, total number of blocks, total number of inodes, block size, inode size, first inode etc. The second block is the group descriptor, which holds information about the groups. There are many backups of the superblock and the group descriptors on the partition. The positions of these backups are determined by the size of the disk, but they are commonly found in group nr 1, 3, 5, 7 and 9. If the superblock and group descriptor are present in a group, they always occupy the two first blocks. In each group the block bitmap and inode bitmap are present. They are located immediately after the superblock and group descriptor if present, or at the two first blocks in the group. The block and inode bitmap keeps information about which blocks and inodes that are used in one group. The next blocks are used by the inode table. An inode entry has a size of 128 bytes, and in a system with 4 Kilo bytes blocks, one block contains 32 inodes. The amount of inode table blocks in each group is determined by the size of the disk and the block size. When using a block size of 4 Kilo bytes, there will be approximately 500 inode table blocks. The exact number is found in the superblock.

2.2 Sector number versus block number

The hard disk needs a reference to where it should read or write. This reference is the sector number. The size of one sector may vary, but a common size is 512 bytes. Each disk may be partitioned into smaller logical disks. A logical disk acts as a physical disk in the way a user sees it, but is actually only a part of one bigger physical disk. Again the operating system needs a reference to where the disk should perform a read or write operation. The block number acts as this reference. The block number tells the position of a file in relation to the currently active partition. This means that the first block of one partition is block number zero, but the sector number for the first block might be 800. The size of one block may vary from system to system, but a common value is 4 Kilo bytes. Thus, one block consists of eight sectors.

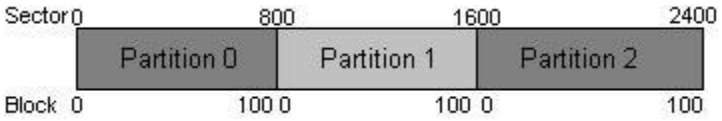

Figure 1 Relation between sector, block and partition

As seen on Figure 1 the block number applies for one partition, but the sector number applies for the entire disk.

2.3 Map sector to block

To map a physical sector to a logical block, you need some information about the disk architecture. The sector number corresponds to the physical disk, and the block number corresponds to the Linux partition on the disk.

To calculate the block number, four values must be present.

- The sector number where the file physically is written on the hard disk.
- The first sector of the partition where the file was written.
- The size of one sector.
- The size of one block.

The sector size and start sector of a partition can be found with the **fdisk** program in Linux. The block number is then calculated by

$$\text{Block} = \frac{(\text{sector} - \text{start sector})}{(\text{sector size} * \text{block size})}$$

Figure 2 Sector number to block number.

3 Virtual Machines

A Virtual Machine (VM) is a software component which simulates a real computer. The first VM was developed by IBM [13] in 1967, as a result of their need to disable certain modules in the real computer, while the operating system was running.

The VM has its own virtual BIOS and hardware, which the operating system inside the VM uses as if it was real hardware. The VM interprets and forwards information from the operating system in the virtual machine to the computers "real" operating system, which does the real system calls.

One useful application of a VM is the advantage that it easily can be turned off, and its status at shut down can still be checked. The state at turn off can be copied and modified if necessary. When testing operating system this advantage can be used to track down where an error occurred. Today there are developed several different virtual machines, such as Plex86, Bochs, and VMware.

VMware [10] was founded in 1998, and in 1999 they released their first product VMware Workstation. VMware Workstation was developed as an industry standard, and soon became a very popular virtual machine. VMware can be used by both Linux and Windows operating systems, and are currently supporting Windows, Linux, Novell Netware and FreeBSD as the guest operating system. The advantage of VMware is that software developers can check if newly developed software can run on different platforms. When using a VM the risk of severe damages is removed, since tests are conducted in a virtual environment.

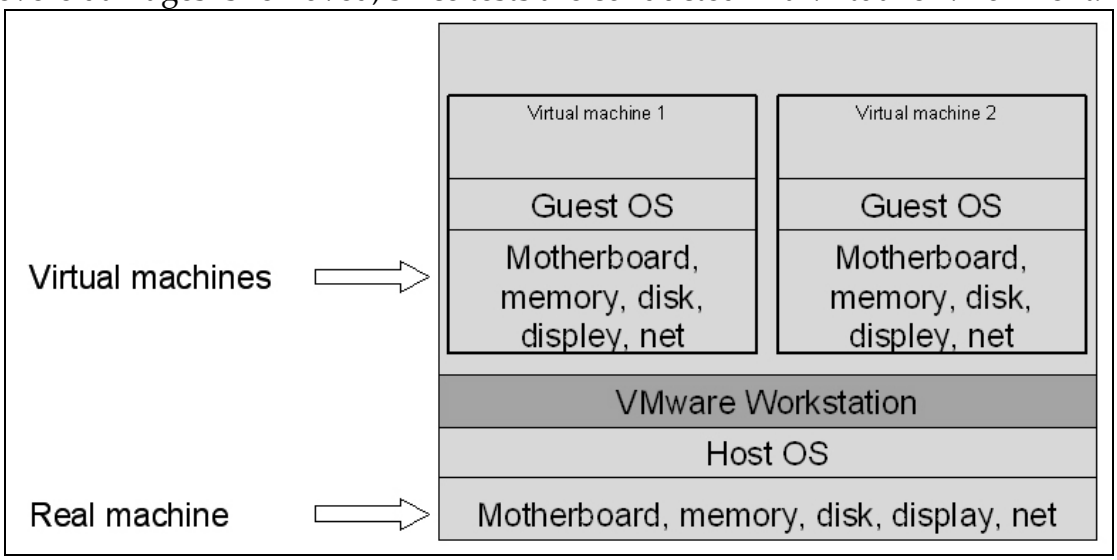

Figure 3 VMware Architecture

3.1 VMware

The virtual machine shall not be aware that there exists any form of control on its content. This means that there shall not be any running processes that may indicate that the currently logged in user is monitored. This makes it necessary for the integrity check to run on the host's operating system.
The virtual operating system acts as an ordinary operating system. When the virtual operating system wants to do a read or write to a disk, it tells the IDE driver what it wants. When a file is going to be written to the disk, the file is transformed to a sector number and a buffer, which is the content of the file. This request is then sent to the disk, in this case the virtual hard disk. VMware takes these calls and forward them to the host's operating system, since only the host's operating system can perform input and output operation to the physical disk. In the host's operating system read and write requests are processed through the IDE driver.

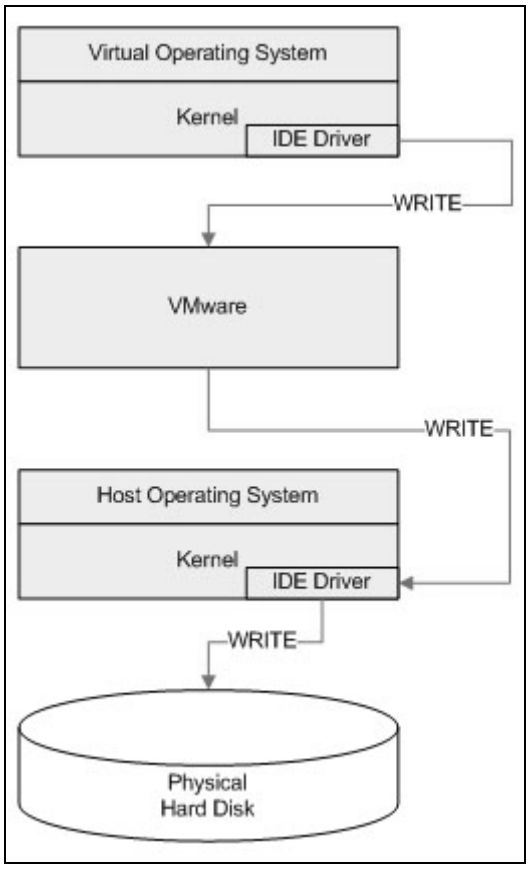

Figure 4 Write call from guest operating system

3.2 Raw disk / Virtual disk

With VMware it is possible to use either a virtual disk or a physical disk. The virtual disk is a file where VMware stores the file system. These disks are flexible and the file containing the disk information will grow according to how much that are written to the virtual disk. It is easy to install an operating system in to a virtual disk since the hard disk does not need to be repartitioned. Physical or raw disks use an existing disk or partition to store the data. The virtual machine is responsible for the data structure on this disk.

3.3 Mounting VMware disks

There are several different methods to connect to the virtual machines file system. VMware is delivered with different programs which can be used to connect to the guests file system. **vmware-mount.pl** is a Perl script which can only be used for connection to the file system, when the VM is turned off. This script enables read and write commands to the file system. **Vmware-loop** is another program that uses a network block driver to connect to the file system. The file system is mounted in a read only mode, because writing to the file system while VMware Workstation is running, would make the file system inconsistent. If the host alters the block bitmap or the inode table by writing a file, the virtual machine would not be aware of this alternation, because it only reads the inode table and block bitmap at start up. There is a problem with both **vmware-loop** and normal mounting of the virtual machine's disk. When the virtual machine writes a file to the disk, the mounted file system is not updated. The reason to this is that the host operating system does not know that the disk is being updated. To see the changes, the host has to remount the virtual machine's disk. Thus the integrity control will not be able to find any integrity violations until the disk is remounted. The vmware-mount.pl and vmware-loop might be used on both a virtual and a raw disk. If a raw disk is used, it can be mounted normal by the mount command in Linux. This method is quicker than using the vmware-loop.

3.4 VMware Networking

VMware Workstation allows three different types of networking, bridged, NAT, and host only.

3.4.1 Bridged Networking

With bridged networking, VMware use a virtual bridge and connects to the host Ethernet adapter. The client gets its own IP-address on the network and acts as a normal computer. This is useful if the client is going to serve different networks services, such as web server or FTP.

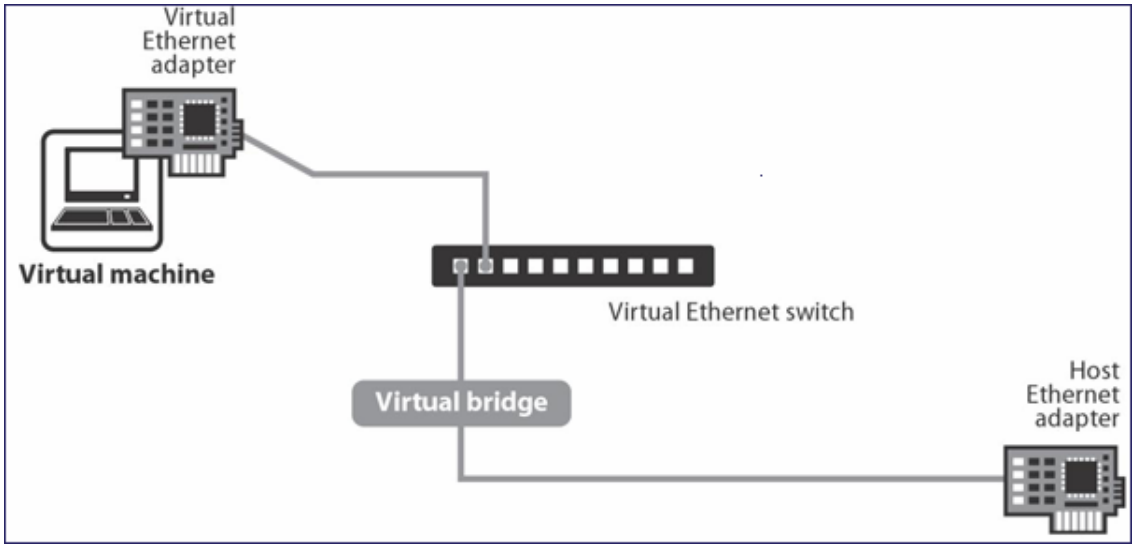

Figure 5 Bridged Networking with VMware [12]

3.4.2 Network Address Translation

With Network Address Translation (NAT) [Figure 6] the virtual machine gets a private IP-address instead of a unique IP-address in the network. The virtual machine get access to the network through the host's IP-address and the host are forwarding packages that are going in and out from the virtual machine. Computers on the network can not initiate connections to the virtual machine, because only the host is visible on the network.

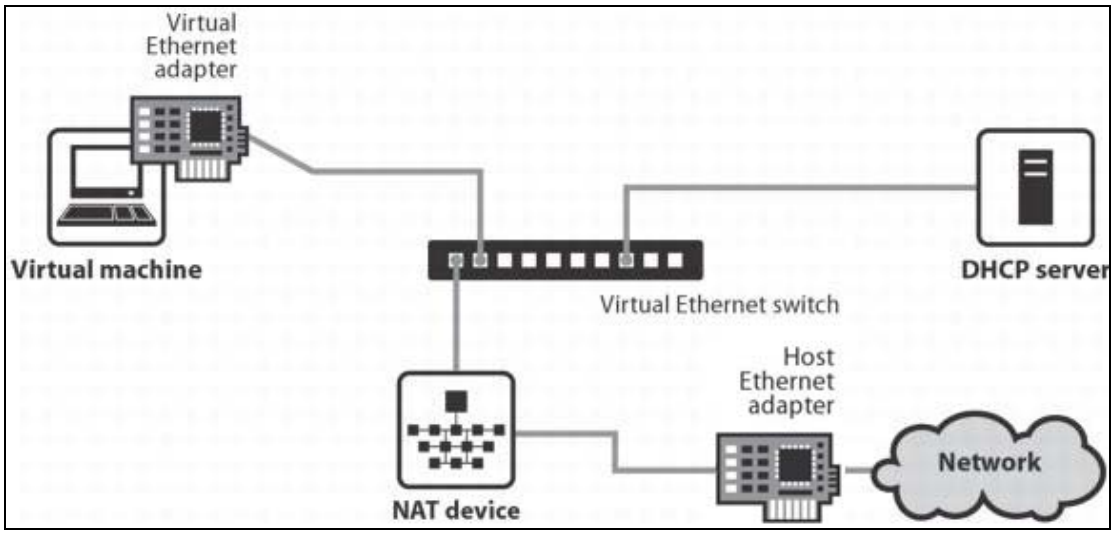

Figure 6 Network Address Translation with VMware [12]

3.4.3 Host only networking

With host only networking VMware sets up a network between the host and the virtual machine only. It uses a virtual network adapter that is visible only to the host and the virtual machine will not be able to access the rest of the network. This is useful to set up a virtual network for development with several virtual machines.

4 Intrusion detection

There are two main types of Intrusion Detection Systems (IDS), Network IDS (NIDS) and Host IDS (HIDS). NIDS is a proactive type of intrusion control which will try to detect whether someone tries to send some malicious code that can be used to break into the network. The benefits with a NIDS are that there is only need for one per network. The HIDS reacts in another way, it detects if an attack succeeds and has lower false positive rate.

There are some problems with a HIDS, if someone gains root access they can discover or disable the IDS. To prevent this, the IDS can be hidden, an alarm could be raised, or the IDS could fight back against the intruder. Another problem is that the HIDS has to be installed on every single computer on the network which is going to be protected, this could be hard since different computers have different configurations and the IDS must be configured for each computer.

4.1 Integrity Control

If someone manages to break through the IDS there should be a second perimeter of defence to detect if someone gains access to the system. On a Linux computer there is many log files which programs and the system reports to when something is done in the system. An intruder can gain access to these logs and manipulate them in such way that no one can see that the security was violated. As a second perimeter defence, integrity control software can be installed. Integrity control does not prevent users from reading or change files, but modified files can be discovered.

4.1.1 File integrity

On a computer there is a huge amount of files and folders. Some of the files can be used and modified by normal users, and others can only be accessed by the administrator. Some files should not be touched at all. To protect these files there is used permissions to control who can use which files. In a standard operating system, an administrator can use his privileges to change files. Some of these changes are often done by a mistake and some are done by purpose. To detect the changes in the file system integrity checks can be used. There are simple ways to check integrity of files.

```
#!/bin/sh

ls -laRi / >/home/files
cd /usr/bin
ls | xargs -I md5sums {} > /home/sums

diff /home/files /home/files.old
diff /home/sums /home/sums.old
```

Figure 7 Simple integrity check

This small program [Figure 7] [05] generates hash sums of the files and compares these sums to old sums stored in a file. If the two files differ, one or more of the files has been compromised and the system administrator should be alerted. Something has happened to a least one of the files we are checking. But this small program is not usable in large systems. On a large system it is better to use a program designed for integrity checking. There are several programs on the market which can be used for integrity checking, for example Tripwire and AIDE.

4.1.2 Fake reports

There are still some problems with integrity control. Often integrity software generates a report after the integrity check. An intruder can manipulate the report, one way to do this is to gain access to the mailer daemon and manipulate the email sent to the administrator. The intruder generates a fake report that claims that the system is OK.

4.1.3 Attack between checks

Another way to fool the integrity check if it is not implemented decent is to do an attack between the checks. An intruder may modify some files to do his malicious work. When he is finished he may restore the original files again. The integrity check will then not be able to detect that these files has been modified.

4.1.4 Read only devices

An intruder can gain access to the integrity database and manipulate it. He might update the database and commit the changes he has done to files. To prevent this, the database can be placed on a read only device, such as a write protected floppy disk or a CD-ROM. The intruder will then be able to access the database but he will not be able to manipulate it.

4.2 Tripwire

The first version of Tripwire was developed in the late 1980's by Gene Kim and Eugene Spafford [03]. It was released in 1992 and in 1997 Tripwire Inc began to license the software from Purdue Research Foundation. In October 2000 Tripwire Inc. released the source code for Linux under the GNU Public License (GPL). There are minor differences between the free version of Tripwire and the commercial version, but as an integrity control the free version is good enough.

Tripwire uses a database where a snapshot of the secure file system is stored. When integrity checking the file system, Tripwire looks up in the database and checks the files. If any violation are found Tripwire will generate a report which an administrator can examine to find out if these changes where normal or not. Tripwire supports different message-digest algorithms for integrity checking such as MD5 sums, HAVAL [04], and CRC-32. Tripwire also checks file attributes, such as permissions, owner, group, inode, creation time, and accessed time. This prevents the attack between checks problem.

With Tripwire it is possible to encrypt the database to prevent intruders to access the database and modify it. The database, configuration, policy and report files are encrypted with an El-Gamal 1024 bits signature which protects them from being modified. It is possible to do an integrity check on the database itself as an extra protection.

Tripwire generates reports after a security check of the system, and it is possible to send these reports by email to an administrator which examines the reports and eventually correcting the errors in the system. It is possible to configure Tripwire to just send reports if there where some violations on the system.

4.2.1 Modifications of Tripwire

When testing Tripwire is was discovered that it would not send an email when checking a single file form command line. We had to modify the source code to allow this.

4.3 AIDE

AIDE (Advanced Intrusion Detection Environment) [09] was developed by the Finnish students Rami Lehti and Pablo Virolainen. AIDE is distributed under the GPL and therefore people can freely use and develop the software. As Tripwire, AIDE supports different cryptographic options to check a file such as MD5, SHA1 and HAVAL among other types, and it is easy to implement more algorithms. It also provides check of file attributes. AIDE differs from Tripwire

because it cannot encrypt its own database and neither supports email notification to system administrators.

4.4 Tripwire or AIDE

We first deployed a model with AIDE because it was stated that it was faster than Tripwire [06]. In tests early in the project we also concluded this, but we had done a mistake. We did not test on a large enough databases. When integrity checks on small databases Tripwire and AIDE had nearly same response time, and when doing an integrity check on a single file AIDE was slightly faster. But when testing on larger databases AIDE was faster on doing an integrity check on the whole system, but Tripwire was much faster on single file checking against the database.

5 Prototype

5.1 Introduction

We have looked upon three alternative solutions to monitor which files that are currently being modified on a disk.
VMware may run in different modes, such as undoable and persistent mode. When using undoable mode, VMware creates a snapshot of the Client at the start up and a redo log file, which shows which file that has been written or deleted, is created. If this file is read by the Constroller, it would know which files that are going to be modified. A problem with this solution is that the files are not written to the disk before the snapshot is committed, which happens when the Client is powered off. This is not convenient since the virtual machine will run as a server, and should thereby rarely be turned off. Another solution is to modify the VMware's virtual IDE driver. If this can be done, the Controller will know which blocks that are currently being updated. But VMware is not an open source project, and thus it is difficult to get access to its source code. The last solution we have looked upon is to modify the Controller's kernel. Since VMware must use the Controller's IDE driver to write to disk, the Controller will be aware of which sectors and blocks that are currently being written.

5.2 Write operation

The Client and Controller behaves equal when it comes to writing files to a disk. Although a big difference is that the Client writes to VMware's virtual hard disk, and the Controller writes to the physical hard disk.
When a file is written on the Client [Figure 8], the file is sent to its IDE driver. The IDE driver transforms the file to a Request. A Request is either a write or read call to the disk, and contains the first sector it will use on the disk, number of sectors it will occupy and a buffer which is the content of the file. This request is then written to the disk, in this case the virtual disk. VMware forwards this write request to the Controller, who sends it to the IDE driver, and the request is written to the physical disk.

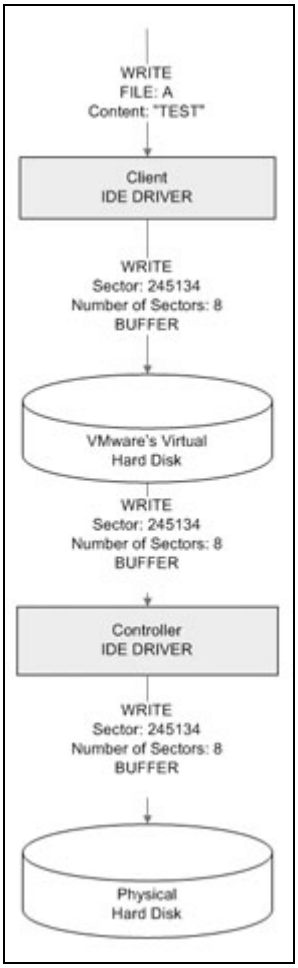

Figure 8 A write operation initiated by the Client,

5.3 IDE driver

To analyze the sector numbers that are written to disk, the IDE driver must be modified. The ide-disk file was chosen [07], since this is at a low level in the IDE driver. The function __ide_do_rw_disk was modified to output the first sector number and how many sector number the write calls occupies. To generate this output an error message by printk was chosen. In Linux there are different categories of error messages, and you may decide where these messages are displayed. The KERNEL_EMERGENCY error is triggered when the kernel has been corrupted. This error occurs rarely, and would thereby fit well as output message from the IDE driver. These types of messages are then displayed in a First In First Out file, or named pipes as they are called in Linux. The advantage with this kind of files is when a line is read, the line is removed.
The output of the IDE driver is printed to a file, and it must be ensured that the source of the output is the Client. If this is not done correctly, there will be an infinity loop in the IDE driver, since the printk message is written to a file, and will as any other file be processed in the IDE driver. If the Client is running on the same physical hard disk as the Controller, the output may be constraint by using the start sector and end sector of the Client. If the Client is running on its own disk, you may constraint this by checking the hard disk name, such as /dev/hda.

5.4 Modifying the IDE driver

When the IDE driver is modified, it is important that the modification does not demand too much processing, since the IDE driver is an important part of the system. If the modified driver demands more processing time than before, the system performance will decrease. It is important that the modification does not interfere with the originally code in such a manner that the system becomes unstable. The IDE driver has been modified to output the block number instead of the sector number. This calculation should take place in the prototype rather than in the IDE driver. But it enhances the human readability, which has been important in the research work to understand how the IDE driver behaves.

5.5 Kernel output

The output from the IDE driver must tell which file that is currently being written. After analyzing the structure variable Request that is passed to the __ide_do_rw_write function, it was discovered that the sector number would give the block number for the currently written file. From the block number the inode and the path can be retrieved. When only one file was written, the path name can be found with this method, but if multiple files were written in one directory, only the first file was found. By analyzing the Request variable further,

the number of sectors variable was investigated. This variable contains the number of sectors the write operation occupies. When multiple files are written in the same directory, and the files are placed immediately after each other on the disk, the number of sectors must be used to find the succeeding files. The number of sectors must be divided by the number of sectors one block occupies in order to find the number of blocks needed by the write operation. To find all written files, each of these blocks must then be mapped to inodes. If file A and file B are written to disk and both are smaller or equal to the block size, the output from the IDE driver will be the start sector of file A and the number of sector this write calls needs, which is 16. If file A is written to sector X, then sector X + 8 is the first sector of file B. If a file is larger than the block size, all of its blocks are mapped to the same inode.

5.6 Filtering blocks

Not only files are written to the disk. The inode table, inode bitmap and block bitmap are also written. These blocks can obviously not be mapped to pathnames and must be filtered out. The block and inode bitmap are always at the start of each group, and occupies one block each. The inode table uses approximately 500 blocks and is succeeding the bitmaps blocks. The superblock contains information about blocks that can be filtered out, such as the number of the first valid inode. The blocks that are bound to invalid inodes can then be filtered out. When setting up the filter the architecture of the disk must be known. By using the program dumpe2fs the needed information can be found.

5.7 Persistent mode versus non persistent mode

VMware has an option for persistent mode, meaning when a new file is created it is immediately written to disk. Ideally this mode would fit well in our system, but it creates a big problem. This is due to the fact that the inode table is not updated until the disk scheduler tells the virtual operating system to write its buffer to disk. When the buffer is written to disk is dependent on time and how fast the buffer is filled up. From a file has been saved and until the buffer is written to disk, the elapsed time is approximately 30 seconds.
Since the inode table is not updated, the block cannot be mapped to an inode, and the path would be impossible to find. Thus using persistent mode will cause the prototype to try to map a block to an inode before the inode is written, and thereby causing unnecessary disk reading.
When the non persistent mode is used, the sectors may be intercepted when the scheduler tells the operating system to write its buffer to disk. This operation takes place some time after the actual write operation, and the integrity check

will not executed in real time. Although the advantage is that the inode table is written at the same time, and the prototype will find the inode at once.

5.8 Scheduler

The operating system has a disk scheduler that triggers when blocks are written to the disk. Writing to disk is a time consuming operation, and the operating system waits to see if there are more sectors that can be written in the same operation. The scheduler organizes the sectors in such an order that it uses less time to write them than the original system calls would have used, and thereby achieving a more time efficient writing procedure. In this project disk scheduling occurs two places; in the Client's and in the Controller's IDE driver. The virtual machine has an option for writing files immediately to the disk, the persistent mode, but the inode table is not written at the same time as the file. This makes it impossible to map "new" blocks to inodes before the inode table has been written. If there could be a process executing the sync command on the Client, this problem would have been solved, since sync forces the buffer to be written to disk.

5.9 Mapping sectors to path

When a file is written to the disk the IDE driver tells which sector and how many sectors the file will occupy. To identify the written file, the block number must be calculated from the sector number.

$$\text{Block} = \frac{(\text{sector} - \text{start sector})}{(\text{sector size} * \text{block size})}$$

Figure 9 Sector number to block number.

This calculation is done by the expression in [Figure 9]. When the block number is identified, the block's inode must be found. All inodes in one group are stored in the group's inode table. To find the inode for one block, all inodes in the inode table must be analyzed to check whether they have a reference to this block number or not. When using a block size of 4 Kilobytes, there are approximately 500 inode table blocks in one group. In each inode table block there are 32 inodes, making the total number of inodes in one group to approximately 16000. Because of the large amount of inodes that must be analyzed, it is important that the program that searches for the inodes are fast. We have looked upon debugfs to do this mapping process. Debugfs does also have a possibility of doing the inode to path mapping process.

5.10 Debugfs

Debugfs is an open source program bundled together with e2fstools, written by Theodore Ts'o. The current version of Debugfs is 1.35, released in February 2004. Debugfs is a tool for debugging an EXT2 or EXT3 file system. Within Debugfs there are among others, tools for mapping between block to inode and inode to path. The information from the superblock may also be viewed in order to examine the architecture of a partition. The code for mapping between block to inode are found in icheck.c, and the code for inode to path mapping are found in ncheck.c. Icheck.c and ncheck.c uses many functions from other header files in debugfs, and the ext2fs library. All these header files are poorly documented and thereby difficult to interpret.

5.11 Modified and new files

When a modified file is written to disk, two scenarios may occur. The file uses either the same block as before the update, which happens rarely, or it uses a new block. Since the file mainly uses new blocks, the inode table needs to be refreshed. Debugfs reads the inode tables at start up and makes an inode cache. This operation takes some time, and should therefore not take place to often. A solution to this problem is to update the inode cache only when a block to inode operation does not return a valid inode. This requires the filter to work properly, and only blocks that belong to files are run with the debugfs tool.

5.12 Concept of prototype

The prototype [Figure 10] must read the FIFO file generated by the IDE driver. When the calculation from sector to block is finished, the filter must be applied. If the block is valid, the process of mapping a block to inode is started. If this process fails, the block must wait for the inode table to be updated before the inode can be found. When the inode is found the process of mapping inode to path is executed, and the path for the sector is outputted.

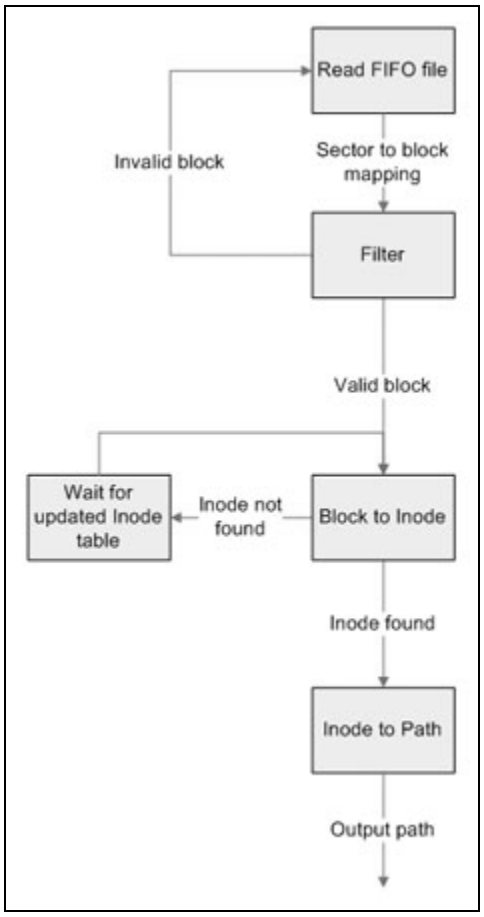

Figure 10 The concept of sector to path mapping

5.13 Solution #1

To map a block to an inode, a function in debugfs called do_icheck is used. do_icheck's code is found in icheck.c. Some of do_icheck's code has been removed, such as some output code, which have no purpose in the prototype. do_icheck has been modified to return the inode number or zero, indicating if the inode is found or not. If a legal inode is found, another function in debugfs called do_ncheck is used. do_ncheck is found in ncheck.c and maps an inode number to a path. It has been modified to either return the path for the inode or zero. When the inode table is written to the disk, it is important that Debugfs's inode cache is updated in order to find the new inodes. This is done by closing and reopening the file system. Some problems using both icheck.c and ncheck.c have been experienced, and any solutions to these problems have not been found.

5.14 Solution #2

Since using both icheck.c and ncheck.c in the same program caused some problems, another solution was developed. This was to run debugfs with system calls, and pipe the information gained into another FIFO file. When the mapping process from sectors to blocks and the filtering is done, the inode can be found by the system call:

debugfs /devicename –R 'icheck [blocknumber]'

This call returns the inode number, which is piped to another FIFO file. The path can be found by the system call

debugfs /devicename –R 'ncheck [inode number]'

By using system calls you ensure that the file system is up to date, because the inode table is read for each call. Since both icheck and ncheck must read the inode tables at each call, these operations will take some time. Typically between 1 to 10 seconds.

5.15 Solution #3

Executing solution #2 takes time, and due to this another solution was developed, which was a hybrid of solution #1 and #2. In this solution the inode table is only refreshed if an inode is not found. This ensures that for each block that belongs to a file, the inode is found. The icheck.c is included in the solution, and the mapping from inode to path is executed by a system call.
When a block has been mapped to an inode, the inode number is put in a queue. Debugfs's ncheck is used to map inode numbers to paths. Ncheck reads the

inodes table each time it is executed, and it can map many inodes to paths for each call. By keeping the inode numbers in a queue, ncheck executes with all queued inodes. The returned paths from ncheck are kept in a new queue, which is used to generate the system call for Tripwire.
This solution is multithreaded and uses four threads. The first thread executes the system call sync every second, to be sure that no blocks from the virtual machine are buffered in the host. The second thread reads the FIFO file, applies the filter and executes the block to inode function. The third thread executes the system call for debugfs. And the last thread reads the file generated by the system call to debugfs and executes the integrity check. This solution guaranties the finding of a path for each block. A drawback is that it might be a bit slow since the process of mapping inodes to paths reads the inode tables each time it is executed.

5.16 Iptables

To prevent the Controller gaining access to the Internet we use a firewall called iptables [08]. Iptables can be used to block connections from certain addresses and ports. When opening more ports than necessary, additional vulnerability is created in the system. In this project the host should not be accessible from the network. Neither should the host be able to access the network, except if the integrity is violated. Precaution where made in the prototype to achieve this criteria. Iptables is a replacement of the old Linux firewall ipchains. Iptables is a part of the Linux kernel, which allows faster executions than a stand alone program.

Iptables makes it possible to specify which programs that are allowed Internet connection. Since we are using bridged networking it is easy to filter the Controller's network traffic from the Client's. With iptables all incoming and outgoing connections from the Client are allowed. The administrator can modify iptables to decide which incoming connections are allowed to the virtual machine, since the Client may have several different servers.

In [Figure 11] there is a sample configuration of the iptables on the host. Following this rule, all incoming connections to the virtual machine will be routed to the Client, and all other connections blocked.

```
Chain INPUT (policy ACCEPT)
target prot opt source           destination
ACCEPT tcp  --  212.125.237.174  anywhere    tcp spt:smtp state RELATED,ESTABLISHED
ACCEPT all  --  anywhere         192.168.1.4
DROP   all  --  anywhere         192.168.1.3


Chain FORWARD (policy ACCEPT)
target prot opt source           destination
DROP   ALL

Chain OUTPUT (policy ACCEPT)
target prot opt source           destination
ACCEPT tcp  --  anywhere         212.125.237.174    tcp dpt:smtp
ACCEPT all  --  192.168.1.4      anywhere
DROP   all  --  192.168.1.3      anywhere
```

Figure 11 Iptables configuration

5.17 Tripwire

Tripwire does not allow sending email reports when checking a single file against the database. We had to do some modifications in the source code of Tripwire to allow this. Tripwire can check several files from command line against the database. If it finds an integrity violation in a file, it generates an email containing a report about which files this is.

6 Results

6.1 Introduction

In this project a vital part was to investigate if real time integrity control of a virtual machine was possible. By going through the IDE driver in the Controller, it is possible to analyze which files the Client is currently writing. When the prototype has identified the path for this file, Tripwire compares the file against its database, and checks if the file is modified or not. If the file is modified, Tripwire sends a mail to the system administrator.

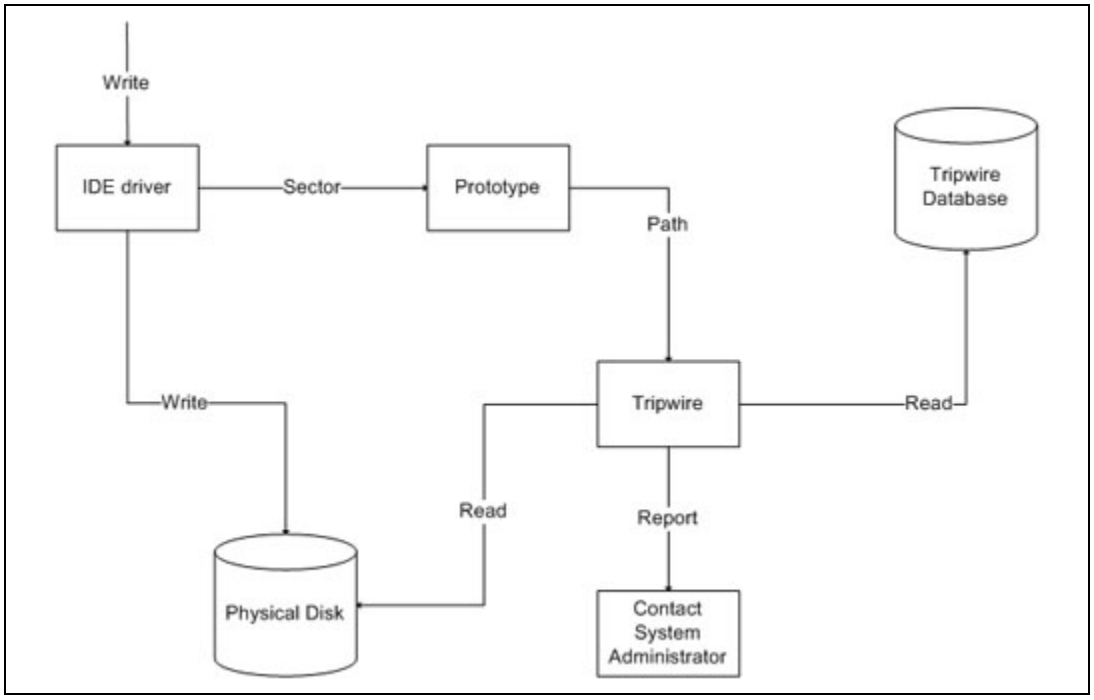

Figure 12 The integrity check

6.2 Performance of the prototype

Since the persistent mode only writes the file to the disk and not the inode table, the prototype must wait for the inode table to be written before it is able to find the path. The inode table is written approximately 30 seconds after the file was modified. How fast a modified file is discovered is dependent on how much activity there is on the disk, since the process of mapping blocks to path demands a lot of disk reading. When a new block is used, the prototype must refresh the inode cache, which uses approximately one to two seconds. The inode is then found in three to four seconds. To find the path for an inode, ncheck needs on the average five seconds.
This means that the prototype will use approximately 40 seconds to find the path for a sector.

6.3 File operation

When a file is created the prototype returns the path for the file and the path for its directory. The integrity check can not detect any abnormality with the new file, but it reports any changes in a directory. If a file is deleted, the prototype returns the file's directory and Tripwire will then figure out which file that is missing. If a file has been modified, the prototype returns the path for the file and Tripwire will then run an integrity check on this file.

6.4 Response Time

There are many factors that affects the response time. The main factor is how much activity there is on the disk. If the virtual machine is reading and writing a lot to the disk, the performance of the prototype will decrease. The prototype needs to do a lot of disk reading to perform the integrity check. It is thereby important that the mapping process from blocks to inodes does not run together with the mapping from inodes to paths. Tripwire's database will also affect the performance of the prototype. If it grows large, the integrity check will be slower. The prototype will also affect the performance of the virtual machine. When the prototype has a large queue of blocks, the performance of the virtual machine will decrease, since they use the same physical disk.

6.5 Performance test of Tripwire and AIDE

We tested AIDE and Tripwire to find out which of the integrity software that where the fastest. We did several testes with both checking the whole database and just checking a single file against the database. All the tests where done on a Linux Red Hat 9 distribution, with a Pentium 4 , 1.6 GHz processor and 1 Gigabyte of memory. There where no other applications running during the tests. During the tests we checked the permission of files and the md5 checksum. As we can see in [Figure 13] AIDE is slightly faster to check small databases but when the database exceeds 5000 files Tripwire are faster.

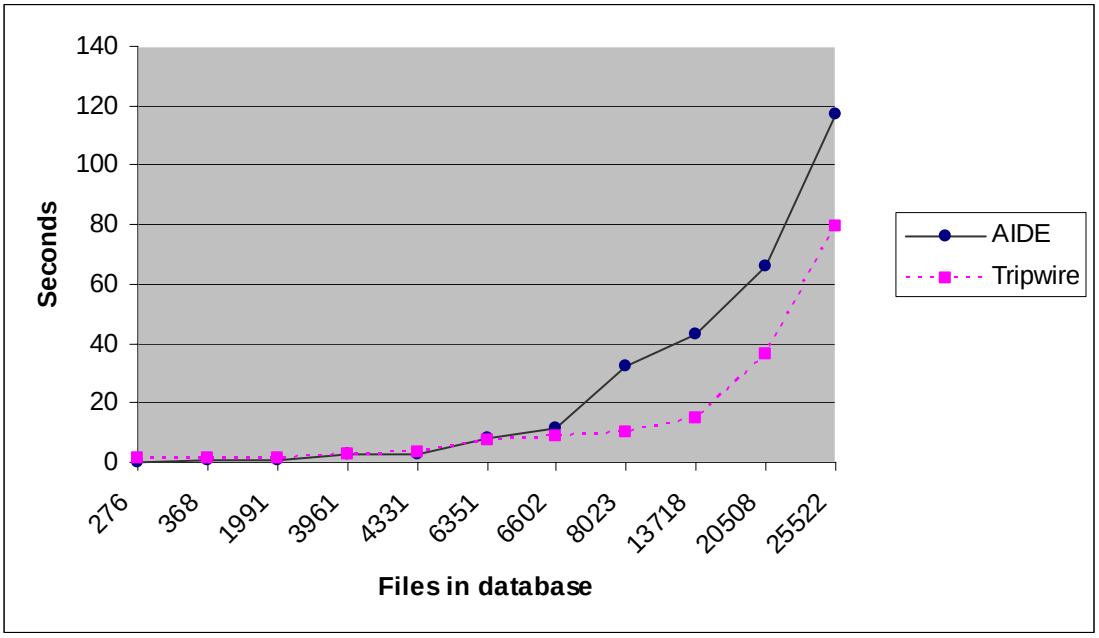

Figure 13 Integrity check against whole database

We also tested integrity checking on one single file against the database [Figure 14] and found that when the database exceeds 10000 files Tripwire is extremely faster. On a database containing 25000 files AIDE uses slightly under 20 seconds but Tripwire uses about 4 seconds. This means that Tripwire is five times faster while checking one single file against the database.

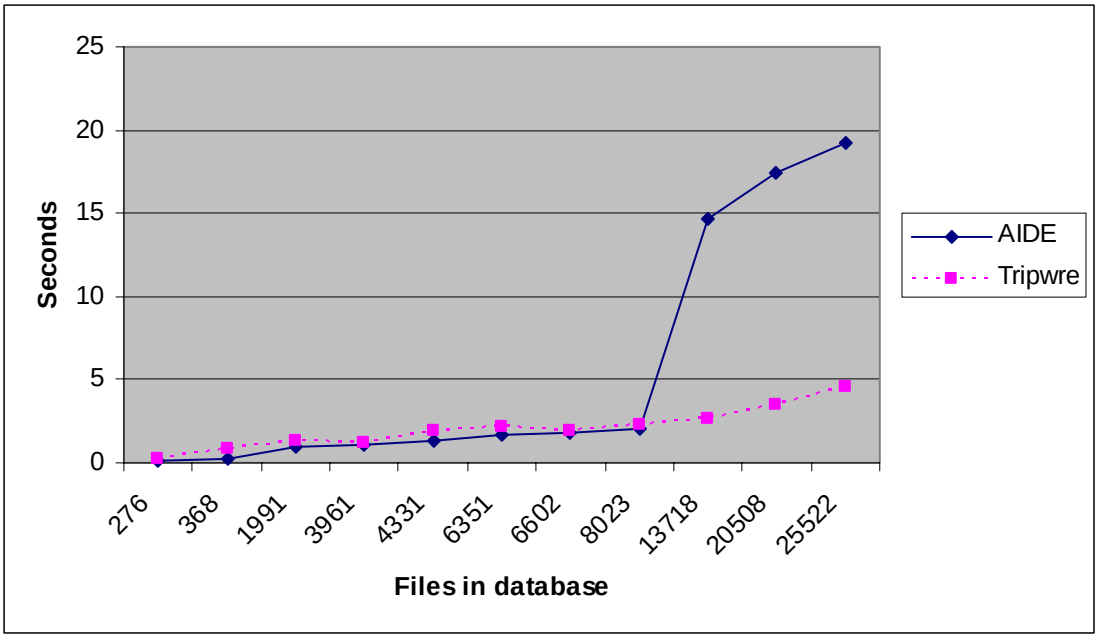

Figure 14 Checking one single file against the database

6.6 Iptables

Iptables are a very powerful filtering option which allows and denies network traffic according to standard filtering options. To prevent the Controller from being contacted through the network interface we have blocked all incoming and outgoing connections except for outgoing SMTP traffic. SMTP allows Tripwire to send a mail to the system administrator if the integrity has been violated. All incoming and outgoing connections are allowed for the Client. This resulted in a secure firewall for this projects purpose. There could be done more filtering on the firewall, but this is dependent of the services the Client provides.

6.7 Mounting file systems

Before an integrity check is performed, the file system must be remounted, since the Controller does not know that the mounted file system is modified. When the vmware-loop was used, remounting the file system uses approximately 1.5

seconds. The normal mount command in Linux used approximately 0.5 seconds, which is about one second faster than the vmware-loop mounting.

6.8 Real time

To detect modified files in real time, the virtual machine has to write blocks and inode tables as fast as possible. Since there should not be any processes running in the virtual machine, and the fact that the persistent mode only writes the file and not the inode table to disk, we need to wait for the disk scheduler to write its buffer, before we can detect a modified file. The buffer is written at a varying interval, but never later than 60 seconds after the file was saved. On the average it is written 30 seconds after the file was saved. This means that the process of mapping sectors to paths, take place on an average of 30 seconds after a file has been modified, and this process uses approximately 10 seconds. The integrity control's speed to detect a modified file depends on the size of its database. With a large database, containing over 25000 files, Tripwire detects modified files in about four seconds. Thus from a file is modified and until the integrity control reports that the file has been modified, the systems uses approximately 45 seconds.

6.9 Area of use

This prototype is meant as an inner defence on a server. Normally there is some kind of outer defence, such as firewall or intrusion detection system (IDS) that will stop an intruder. If an intruder manages to bypass these outer defences and gain access to a server, he will often try to make the computer more vulnerable by modifying some files. When our system is used, the intruder would not be aware that the system has an integrity control, since he is connected to the Client and the integrity control runs on the Controller. When the intruder modifies files on the Client, the Controller will report any changes in files within 45 seconds. The Controller sends an email to the system administrator, and it will be up to him to perform the appropriate action, such as restoring the modified files with the originals.

An important observation is that our system will not deny a user from modifying files. And an intruder might exploit the system during the time from a file is modified until the system administrator has restored the original file.

7 Discussion

7.1 The prototype

Three different solutions of mapping blocks to path have been tested. The selected solution is not the best, since the file system's inode tables must be read for each time the process of mapping an inode to path is executed. This operation should ideally happen only when a modification in an inode table have occurred. In the first prototype, we include both the icheck.c and ncheck.c from debugfs, but there were some compatible problems when both files were used. No problems were encountered when only one file was included, but when both were included Debugfs generated a Segmentation Fault error in different parts of the code. Debugfs source code is complex and poorly documented, which made the debugging process difficult. In the end we had to abandon the debugging and search for another solution. We then developed a system that executed both the mapping from block to inode and inode to path with system calls. This approach worked fine, although the performance was bad since both mapping processes needed to read the inode table for each call.
The solution we ended up with included the icheck.c file from debugfs and executed the mapping from inode to path as a system call.

7.2 Real time

The project aimed to detect modified files in real time. From the view of the Controller, the integrity check happens in real time, since it reacts as soon as a file is written to the disk. Seen from the point of the Client the system is not in real time, because approximately 45 seconds is needed from a file is written until it is detected as modified. Since we got the constraints that there shall not be any running processes which may indicate to a user that he is monitored, the system we have developed is as near real time as you may get.
The system uses some time before a block can be mapped to a path, due to the fact that the inode table is not written at the same time as the file when using persistent mode. This could be solved if a process in the Client had executed the sync command often. If this process had been running, the inode table would have been written at the same time as the file, and our prototype would have found the path as soon as the inode tables had been refreshed. And thereby the prototype would have detected a modified file at approximately 30 seconds earlier than it does now.

7.3 Difficulties

A big problem with this project has been the absence of useful documentation of the IDE driver. Because of this, much time has been used to read the IDE driver's source code. This code is complex and poorly documented, and it has been difficult to understand the behaviour of the IDE driver. An important tool in the research process of the IDE driver has been the Dumpe2fs, which displays the superblock and the group information for a file system. By using different outputs from the Request variable, together with Dumpe2fs, has been vital to understand how the IDE driver behaves, such as when a file is written and when the inode table is updated.
Another problem has been the Segmentation Errors in Debugfs, when both the icheck.c and ncheck.c files where included. The error happened at different parts in both files, and we could not find any reason for the error. Although when they where used separately, they both worked fine.

7.4 Further work

There are possible to make the prototype faster in detecting integrity violations. In the prototype the icheck.c and ncheck.c compatible problems should be solved. This would save approximately five seconds. If the sync command cannot be executed on the virtual machine, the disk scheduler in the virtual machine should be modified. If the disk scheduler writes the inode table at the same time as the file is written, when using persistent mode, approximately thirty seconds would be saved. Although modifying the disk scheduler can be difficult, since there is not much documentation about it. If the compatible problems could be fixed and the disk scheduler modified, the prototype will use approximately 10 seconds to detect that a file is modified.
When there is a great deal of read and write operations on the disk, the performance of both the virtual machine and the prototype will decrease. One solution that might increase the performance of both systems is to add a new physical disk. By modifying the IDE driver to write to both disks, the new disk will be a perfect backup of the virtual machine's disk. By letting the prototype read from the backup disk, it will not interfere with the virtual machines read operations. It is important that both disks have the equal architecture, such as group sizes, block size and total size. This solution has not been tested, but it should work if the IDE driver is modified correctly.

8 Conclusion

In this project we have looked upon the possibility to run an integrity check on a virtual machine as fast as possible after a file's content has been modified. An integrity control is usually rarely executed, maybe only a couple of times per week, and thus it might take a long time from a file is modified until it is discovered by the system administrator.

By using a virtual machine, the Client, there is possible to run an integrity check from the host machine, the Controller, and thereby hiding the integrity check from the Client. This means that the Controller must have access to the Client's disk and the file's that are being written. Since the Client runs inside the Controller, all physical hardware is controlled by the Controller. To be able to detect which file that is being written from the Client, we have proposed a solution that uses approximately 45 seconds from the file is written until the Controller has detected that the file's content is modified. The solution is based on analyzing the Client's write requests, through the Controller's IDE driver.

This system will not deny a user from modifying a file, but it is only monitoring the file's integrity and sending a report if the integrity is violated. The system administrator must perform the appropriate action to deal with the integrity violation.

Because our system uses a virtual machine, it has been important that a user does not have access to the Controller. We have proposed a solution that uses iptables to deal with connection control. All traffic are allowed to the Client, but only outgoing SMTP traffic are allowed from the Controller, which enables mail sending.

We have proposed some action that will make our system faster. The most important suggestion is to modify the Client's disk scheduler to write the inode table at the same time as the file is written. This modification would save approximately 30 seconds from the total time of detecting a modified file.

References

[01] Silberschatz Galvin, Operating System Concepts Fifth Edition, John Wiley & Sons, INC, 1999

[02] EXT2 file system,http://kris.koehntopp.de/inkomploehntopp/01963.html [Accessed February 06, 2004]

[03] Tripwire history, http://www.sao.org/newsletter/companies/woody07-01.htm
[Accessed March 12, 2004]

[04] Y. Zheng, J. Pieprzyk and J. Seberry: HAVAL -- a one-way hashing algorithm with variable length of output, Advances in Cryptology -- AusCrypt'92, Lecture Notes in Computer Science, Vol. 718, pp. 83-104, Springer-Verlag, Berlin, 1993.

[05] Anton Chuvakin, ups and down of UNIX/Linux host-based security solutions, April 2003

[06] AIDE vs. Tripwire, http://www.fbunet.de/aide.shtml,
[Accessed April 12, 2004]

[07] A small trail through the Linux kernel,
http://www.win.tue.nl/~aeb/linux/vfs/trail-3.html
[Accessed February 01, 2004]

[08] Iptables, http://www.iptables.org [Accessed April 20, 2004]

[09] AIDE, http://www.fbunet.de/aide.shtml [Accessed February 6, 2004]

[10] VMware, http://www.vmware.com [Accessed February 15, 2004]

[11] Bridged networking,
http://www.vmware.com/support/gsx3/doc/network_bridged_gsx.html
[Accessed May 04, 2004]

[12] NAT VMware,
http://www.vmware.com/support/gsx3/doc/network_nat_gsx.html
[Accessed May 04, 2004]
[13] The Virtual Machine, http://www.cap-lore.com/CP.html
[Accessed March 05, 2004]

[14] Plex86 x86 Virtual Machine, http://plex86.sourceforge.net/
[Accessed February 08,2004]

[15] Bochs x86 PC emulator, http://sourceforge.net/projects/bochs/
[Accessed February 08, 2004]